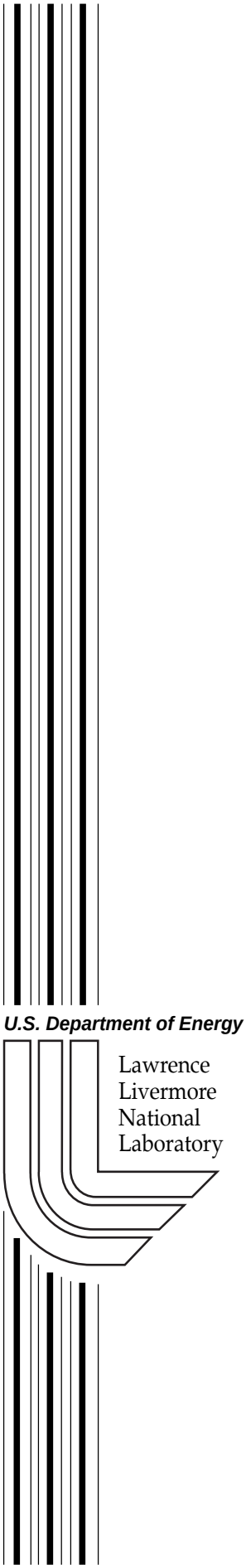
Preprint
UCRL-JC-144818

A Generic Scheduling Simulator for High Performance Parallel Computers

A. B. Yoo, G. S. Choi, M. A. Jette

This article was submitted to Los Alamos Computer Science Institute Symposium, Santa Fe, NM
October 15 - 18, 2001



August 1, 2001

A Generic Scheduling Simulator for High Performance Parallel Computers*

Andy B. Yoo[1], Gyu Sang Choi[2] and Morris A. Jette[1]

[1]Lawrence Livermore National Laboratory
Livermore, CA 94551
e-mail: {yoo2 | jette1}@llnl.gov

[2]Dept. of Computer Science and Engineering
Penn State University
University Park, PA 16802
e-mail: gchoi@cse.psu.edu

1 Statement of Problem

It is well known that efficient job scheduling plays a crucial role in achieving high system utilization in large-scale high performance computing environments. A good scheduling algorithm should schedule jobs to achieve high system utilization while satisfying various user demands in an equitable fashion. Designing such a scheduling algorithm is a non-trivial task even in a static environment. In practice, the computing environment and workload are constantly changing. There are several reasons for this:

First, the computing platforms constantly evolve as the technology advances. For example, the availability of relatively powerful commodity off-the-shelf (COTS) components at steadily diminishing prices have made it feasible to construct ever larger massively parallel computers in recent years [1, 4]. Second, the workload imposed on the system also changes constantly. The rapidly increasing compute resources have provided many applications developers with the opportunity to radically alter program characteristics and take advantage of these additional resources. New developments in software technology may also trigger changes in user applications. Finally, political climate change may alter user priorities or the mission of the organization.

System designers in such dynamic environments must be able to accurately forecast the effect of changes in the hardware, software, and/or policies under consideration. If the environmental changes are significant, one must also reassess scheduling algorithms. Simulation has frequently been relied upon for this analysis, because other methods such as analytical modeling or actual measurements are usually too difficult or costly. A drawback of the simulation approach, however, is that developing a simulator is a time-consuming process. Furthermore, an existing simulator cannot be easily adapted to a new environment.

In this research, we attempt to develop a generic job-scheduling simulator, which facilitates the evaluation of different scheduling algorithms in various computing environments. The following are our design objectives for this generic simulator.

1. Accept descriptions of varied workloads for a wide range of computing environments.
2. Provide an easy-to-use interface for description of the scheduling policies being evaluated.
3. Accurately calculate the overhead induced by various scheduling algorithms.
4. Accurately model a variety of machine architectures.

*This work was performed under the auspices of the U.S. Department of Energy by University of California Lawrence Livermore National Laboratory under contract No. W-7405-Eng-48.

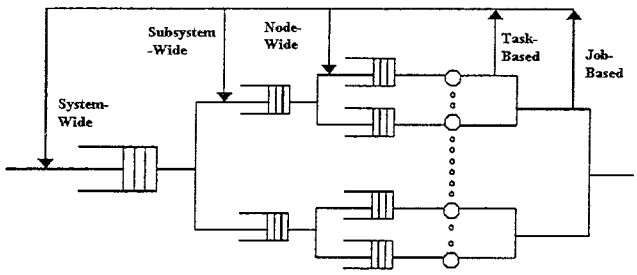


Figure 1: Queuing network representation of the M^2 framework.

2 Design and Implementation

Our generic simulator is based on the Multitasking and Multiprogramming (M^2) framework [6]. As shown in Figure 1, the M^2 represents a closed-end queuing network and consists of three levels of queues: system-, subsystem-, and node-level. The system-level queue would typically represent all batch jobs to be executed on an assortment of computers or cluster of computers. The subsystem- level queue would typically represent cluster-level scheduling such as IBM's LoadLeveler [5] assigning tasks of a parallel job to specific nodes in the cluster. The node-level queue would represent the tasks being assigned to compute nodes. There can be a variety of distinct subsystem and node-level queues.

A job scheduling algorithm is specified by three parameters: scheduling unit (job- or task-based) which defines scheduling granularity, scheduling width (system-, subsystem, or node-wide) that represents the queue level to which a preempted scheduling unit returns, and partitioning mechanism (static or dynamic partitioning) that specifies how the system is partitioned to allocate incoming jobs. Using these three parameters, the M^2 framework provides a very powerful tool that can be used to represent various scheduling algorithms on different computers. An interested reader should refer to [6] for detailed description of the M^2 framework.

A major drawback of the M^2 representation is its assumption of a first-come first-served (FCFS) scheduling policy in managing queues in the system. This shortcoming makes direct application of the M^2 framework difficult in the development of a generic simulator. In developing this generic simulator, we have extended the M^2 framework for the users to be able specify different scheduling algorithms for each queue (or a group of queues).

Our generic simulator accepts three input categories: job characteristics, system characteristics, and job scheduling information. There is a great deal of flexibility in specifying job characteristics. There are predefined attributes such as job inter-arrival time and job execution time. These may be augmented with user-defined job attributes, which may be unique to the system such as resource demands, user, group (or other specification for fair-share scheduling), and priority. The distributions of these attributes can be derived using standard or user-defined distribution functions. Alternately, one can supply a workload trace, optionally including user-defined job attributes.

The performance metrics generated by this simulator are job response time and system utilization. Although the current implementation concerns only these two performance parameters, the simulator can be easily extended to include user-defined performance metrics. Each performance parameter is accompanied by its 95% confidence interval. Particularly with heavily loaded systems, longer simulation periods are required to obtain accurate results. By considering the confidence interval provided by the simulator, users can determine whether a longer simulation period is needed.

System overhead can vary depending upon the scheduling algorithm being used. Paging overhead caused by employing a time-sharing scheduling is a good example of such an overhead. Another good example is refreshing a cache when context switching occurs. Although the overhead induced by scheduling algorithms has become a critical issue, it has been largely ignored by existing simulators [2, 3]. A novel feature of our generic simulator is its ability to calculate the overhead incurred by employing specific scheduling algorithms. This provides both more accurate results overall and a clear identification of the measured overhead into its performance report.

A stream of jobs is generated by using the input generation method defined by the user. The simulation continues until the predefined number of

jobs finish their execution and depart the system. When a job completes execution, two performance parameters are updated.

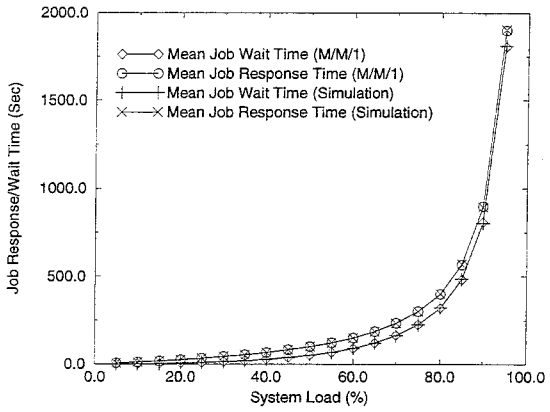
The core of our generic simulator lies in its mechanism to manipulate the network of queues. When a job first arrives at the system, it joins the system queue. When the job is assigned to a subsystem, it enters an appropriate subsystem queue. The subsystem queues can be static or generated dynamically depending on the given partitioning mechanism. Finally, the tasks of the job are assigned to a node-level queue, which manages the processors, cache, and memory within a node.

User specifies a set of scheduling algorithms for each queue in the system. The set of scheduling algorithms for a queue consists of a primary scheduling algorithm and secondary scheduling algorithms. The primary scheduling algorithm is invoked on certain events like job arrivals and departures by default. For each secondary scheduling algorithm, a distribution function, which is used to calculate the interval between (user-defined) events, is specified. The secondary scheduling algorithms are used to capture the occurrence of certain events that are specific to the system being simulated. That is, the secondary scheduling algorithms are used to model scheduling activities that have to be performed when certain events occur. When a scheduling (primary or secondary) algorithm is invoked, the jobs or tasks that are in the queue with which the scheduling algorithm is associated are rearranged. This design allows the users to simulate various scheduling algorithms under varying conditions easily. For example, a secondary scheduling algorithm could be used to simulate expedited work, system failures, scheduled maintenance periods, etc.

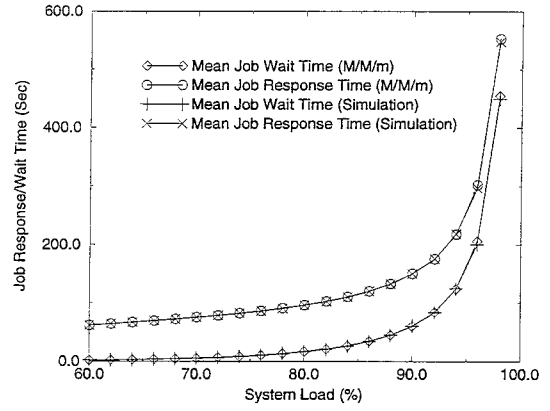
A user writes a scheduling algorithm as a C function and binds it to the queues that the algorithm is defined for. To facilitate this process, we provide users with a set of application programming interfaces (APIs). These APIs are categorized into two classes: attribute retrieval and queue management APIs. The attribute retrieval APIs allow a user to retrieve attribute values of a job. Users use this information to prioritize and order the jobs (or tasks) in the corresponding queue. The queue management APIs can be used to change the order of objects in a queue. Using these APIs, the users can freely describe any scheduling algorithms without knowing the implementation details of the simulator. Typically each function is a small program, but the opportunity exists to define very complex algorithms if so desired.

We have validated our simulator by comparing the performance measurements obtained from the simulator for various loads with those from M/M/1 and M/M/10 queuing models. The results are plotted in Figure 2. In these experiments, we assumed that all of the nodes are identical and incoming jobs require a single node. The results are quite encouraging. The margin of error introduced by our simulator is within 0.2% of the results from the queuing models. We plan to conduct more validation tests for different workloads and scheduling algorithms in the future. We also plan to further study on the overheads caused by different scheduling algorithms and its effect on overall system performance.

In summary, we have developed a generic scheduling simulator for high performance parallel computers. This generic simulator supports standard and user-defined job attributes and generates the job attribute values from different input sources, allowing users to model a wide range of workloads, and produces performance parameters with reliability measures. All overheads caused by scheduling algorithms are considered in measuring the performance parameters. The simulator simulates a queuing network to which users can bound a specific scheduling algorithm written as a C function. A set of APIs is provided for the users to facilitate describing the scheduling algorithms. With these features, this simulator can accurately simulate any scheduling algorithms under various workloads and computing platforms. The simulator does not currently model dynamic events like message passing between tasks closely, but we plan to include this crucial functionality into our simulator in the future.



(a) $M/M/1$ queuing model



(b) $M/M/m$ queuing model

Figure 2: The comparison of results from the simulator and queuing models. Mean job inter-arrival times are fixed to 100 seconds and 10 seconds for $M/M/1$ and $M/M/m$ queuing models, respectively. Mean job service time is the mean job inter-arrival rate divided by a given system load.

References

[1] ASCI Project. `www.lanl.gov/asci`.

[2] H. Bodhanwala, L. M. Campos, C. Chai, C. Decoro, K. Fowler, P. Franck, H. Nguyen, N. Patel, I. Scherson, and F. Silva. A General Purpose Discrete Event Simulator. In *Proc. International Symposium on Performance Evaluation of Computer and Telecommunication Systems*, July 2001.

[3] H. Casanova. Simgrid: a Toolkit for the Simulation of Application Scheduling. In *Proc. International Symposium on Cluster Computing and the Grid*, pages 430–437, May 2001.

[4] Cplant. `http://www.cs.sandia.gov/cplant`.

[5] IBM Corporation. *LoadLeveler's User Guide, Release 2.1.*

[6] C. Yu. *Processor Management Policies for Multiprocessors.* PhD thesis, Dept. of Compuer Science and Engineering, Pennsylvania State University, 1994.